

Cours d'analyse d'image et d'intelligence artificielle.

Chapitre 2: Réseaux de neurones convolutionnels.

Frédéric Richard

Institut de Mathématiques de Marseille

15/10/2024

Master Mathématiques appliquées,
statistique (2ème année),



Apprentissage statistique.

- On dispose d'observations $(x_i, y_i)_{i=1}^n$. On suppose que ces observations sont distribuées selon la même loi de distribution inconnue P que des variables (X, Y) à valeurs dans $\mathcal{X} \times \mathcal{Y}$.
- L'objectif est de prédire Y à partir de X en utilisant une famille \mathcal{F} de fonctions paramétrées par θ : pour $f_\theta \in \mathcal{F}$, la prévision de Y est donnée par $\hat{Y} = f_\theta(X)$.
- On définit une fonction de coût ℓ qui mesure l'erreur de prévision de Y par \hat{Y} .
- On cherche θ minimisant le risque théorique :

$$\mathbb{E}(\ell(f_\theta(X), Y)) = \int_{\mathcal{X} \times \mathcal{Y}} \ell(f_\theta(x), y) dP(x, y).$$

- P étant inconnue, on minimise le risque empirique :

$$E_n(\ell(f_\theta(x), y)) = \frac{1}{n} \sum_{i=1}^n \ell(f_\theta(x_i), y_i).$$

Régression multinomiale.

- Y une v. a. à valeurs dans un espace d'étiquettes (classes) $\{1, \dots, K\}$.
- $X = (X_j)_{j=1}^p$ v.a. à valeurs dans \mathbb{R}^p .
- Modèle de distribution :
 - Paramètres : $\theta = (\theta^{(k)})_{k=1}^K$ avec $\theta^{(k)} \in \mathbb{R}^{p+1}$.
 - Modèle :

$$f_{\theta}(x) = \left(\frac{\exp \left(\theta_0^{(k)} + \sum_{j=1}^p \theta_j^{(k)} x_j \right)}{\sum_{m=1}^K \exp \left(\theta_0^{(m)} + \sum_{j=1}^p \theta_j^{(m)} x_j \right)} \right)_{k=1}^K.$$

- Entropie croisée :

$$\ell(f_{\theta}(x), y) = - \sum_{k=1}^K 1_{y=k} \log (f_{\theta}(x)_k).$$

Interprétation du modèle de distribution.

$$\text{Modèle : } f_{\theta}(x) = \left(\frac{\exp(\theta_0^{(k)} + \sum_{j=1}^p \theta_j^{(k)} x_j)}{\sum_{m=1}^K \exp(\theta_0^{(m)} + \sum_{j=1}^p \theta_j^{(m)} x_j)} \right)_{k=1}^K.$$

Ingrédients :

- Pour $x \in \mathbb{R}^p$ et $w \in \mathbb{R}^{p+1}$, $D(x; w) = w_0 + \sum_{j=1}^p w_j x_j$.
- Soft-max (K classes) : pour $z \in \mathbb{R}^K$, $\varphi(z) = \left(\frac{\exp(z_k)}{\sum_{m=1}^K \exp(z_m)} \right)_{k=1}^K$.

Réseau de neurones :

- Couche d'entrée : $x^{(0)} = x \in \mathbb{R}^p$.
- Couche dense à K neurones nécessitant $K(p+1)$ poids :

$$z^{(0)} = (z_k^{(0)})_{k=1}^K \text{ avec } z_k^{(0)} = D(x^{(0)}; \theta^{(k)}).$$

- Activation "soft-max" :

$$x^{(1)} = \varphi(z^{(0)}).$$

Application à l'image.

Image discrète (en dimension 2) :

$$f = (f[m, n], (m, n) \in \{0, \dots, N_1 - 1\} \times \{0, \dots, N_2 - 1\}).$$

Pour appliquer la régression multinomiale, on vectorise I en définissant

$$x_j = f[m, n], \quad j = (m - 1)N_2 + n + 1,$$

pour $j = 1, \dots, p$ avec $p = N_1 N_2$.

Problèmes et limitations de l'application de la régression multinomiale à l'image $x = (x_j)_{j=1}^p$:

- La dimension de p est très grande (typiquement 2^{16} pour des images de taille standard 256×256). Le nombre de paramètres sur la couche dense (en $O(p + 1)$) est trop important.
- La vectorisation fait perdre l'information spatiale qui est véhiculée par les indices (m, n) de l'image I .

La couche de convolution.

Pour définir la première couche on peut remplacer l'opération $D(x; w) = w_0 + \sum_{j=1} w_j x_j$ par une convolution $C(I; w) = I * w$ de l'image I avec un noyau w à support $\{-M_1, M_1\} \times \{-M_2, M_2\}$: Pour tout $(m, n) \in \{0, N_1 - 2M_1\} \times \{0, N_2 - 2M_2\}$,

$$C(f; w)[m, n] = \sum_{u=-M_1}^{M_1} \sum_{v=-M_2}^{M_2} I[m + M_1 - u, n + M_2 - v]w[u, v].$$

Avantages :

- l'opération ne nécessite que $(2M_1 + 1)(2M_2 + 1)$ paramètres
- et tient compte de la structure spatiale de l'image.
- $C(f; w)[m, n]$ s'interprète en termes de présence du motif w dans le voisinage de (m, n) .

Réseau avec une couche de convolution.

- Couche d'entrée : $f^{(0)} = f$, image de taille $N \times N$.
- Couche de convolutions avec J filtres de support $\{-M, M\}^2$:

$$\forall j \in \{1, \dots, J\}, f^{(1,j)} = C(f^{(0)}; w^{(j)}),$$

$f^{(1,j)}$: carte de caractéristiques (feature map).

- Couche de pooling réduisant la dimension. Par exemple, un sous-échantillonnage d'un facteur Q sur chaque $f^{(1,j)}$

$$\forall (m, n) \in \{0, \dots, (N - 2M)/Q\}^2, g^{(1,j)}[m, n] = f^{(1,j)}[Qm, Qn].$$

- Couche de vectorisation de toutes les sorties $g^{(1,j)}$ en un vecteur x de taille $p = J * ((N - 2M)/Q + 1)^2$

- Couche dense à K neurones :

$$z = (z_k)_{k=1}^K \text{ avec } z_k = D(x; \theta^{(k)}).$$

- Activation "soft-max" :

$$x^{(1)} = \sigma(z^{(0)})$$

Autres formes de pooling.

- Average pooling : filtre moyenne locale de taille $(2Q + 1) \times (2Q + 1)$ et sous-échantillonnage de facteur Q :
 Pour tout $j \in \{1, \dots, J\}$ et $(m, n) \in \{0, \dots, (N - 2M)/Q - 2\}^2$,

$$g^{(1,j)}[m, n] = \frac{1}{(2Q + 1)^2} \sum_{u,v=-Q}^Q f^{(1,j)}[Q(m + 1) + u, Q(n + 1) + v].$$

Interprétation : filtre passe-bas.

- Max pooling : maxima locaux des réponses :
 Pour tout $j \in \{1, \dots, J\}$ et $(m, n) \in \{0, \dots, (N - 2M)/Q - 2\}^2$,

$$g^{(1,j)}[m, n] = \max_{u,v=-Q, \dots, Q} |f^{(1,j)}[Q(m + 1) + u, Q(n + 1) + v]|.$$

Filtre non-linéaire.

Vers un réseau de neurones profond.

Comment multiplier les couches ?

- Succession de U couches convolutionnelles :
pour $u = 1, \dots, U$,

$$\forall j \in \{1, \dots, U^{(l)}\}, f^{(u,j)} = C(f^{(l-1, \iota(u,j))}; w^{(l,j)}),$$

avec des noyaux $w^{(u,j)}$ de filtres de support $\{-M^{(u)}, M^{(u)}\}^2$.

Cela se fait en une seule couche avec des noyaux plus grand.

- Idem s'il l'on ajoute un average pooling.
- Cela n'est pas le cas si l'on ajoute un max-pooling.
- Autre solution : ajout de fonctions d'activation non linéaires :

$$\forall j \in \{1, \dots, J^{(u)}\}, f^{(u,j)} = \varphi^{(u)} \left(C(f^{(u-1, \iota(u,j))}; w^{(u,j)}) \right),$$

où $\varphi^{(u)}$ est une fonction non linéaire (par ex., $\tanh(t)$).

Succession de couches. Pour $u = 1, \dots, U$:

$$\forall j \in \{1, \dots, J^{(u)}\}, f^{(u,j)} = \text{Pool}_{Q^{(u)}} \left(\varphi^{(u)} \left(C(f^{(u-1, \ell(u,l))}; w^{(u,j)}) \right) \right).$$

Nombre de paramètres de la couche l : $(2M^{(l)} + 1)^2 * J^{(l)}$ (taille x nombre de filtres)

Taille de $f^{(u,\cdot)}$ de l'ordre de $(N^{(u)} / Q^{(u)})^2$.

Intérêt de plusieurs couches :

- Hiérarchie de représentations : du fait du pooling, on change d'échelle à chaque couche.
- L'image $f^{(L,\cdot)}$ a une taille de l'ordre de

$$p = \left(\frac{N}{\prod_{u=1}^U Q^{(u)}} \right)^2.$$

Vecteur de dimension réduite pouvant être utilisé comme entrée d'un réseau de régression multinomiale.

Exemple : réseau LeNet (1998).

Classification de la base MNIST (<http://yann.lecun.com/>).

Type	Cartes	Image	Noyau	Pas	Activ.
Entrée	1	32x32	-	-	-
Conv.	6	28x28	5x5	1	tanh
Av. Pooling	6	14x14	2x2	2	tanh
Conv.	16	10x10	5x5	1	tanh
Av. Pooling	16	5x5	2x2	2	tanh
Conv.	120	1x1	5x5	1	tanh
Dense	-	84	-	-	tanh
Dense	-	10	-	-	softmax



Outline

Apprentissage.

Problème d'optimisation.

- Base d'apprentissage : observations $(x_i, y_i)_{i=1}^n$.
- Prédiction de y_i par x_i obtenue sur la dernière couche $\hat{y}_i(w) = f_i^{(U)}$ d'un réseau défini de manière récursive :
Couche d'entrée : $f_i^{(0)} = x_i$ et, pour $u = 1, \dots, U$,

$$\begin{cases} g_i^{(u)} &= S^{(u)}(f_i^{(u-1)}; w^{(u)}), \\ f_i^{(u)} &= \varphi^{(u)}(g_i^{(u)}). \end{cases}$$

où $S^{(u)}$ est une couche dense ou une couche de convolution, $w^{(u)}$ sont les poids de cette couche, et $\varphi^{(u)}$ une activation.

- Fonction de coût :

$$E_n(w) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, \hat{y}_i(w)).$$



Différentielle de la fonction de coût.

$$E_n(w) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f_i^{(U)}) \quad \text{avec} \quad \begin{cases} g_i^{(u)} &= S^{(u)}(f_i^{(u-1)}; w^{(u)}), \\ f_i^{(u)} &= \varphi^{(u)}(g_i^{(u)}). \end{cases}$$

$$\frac{dE_n}{dw^{(v)}}(w) = \frac{1}{n} \sum_{i=1}^n \frac{d\ell}{dz} \left(y_i, \frac{df_i^{(L)}}{dw^{(v)}} \right).$$

$$\frac{df_i^{(u)}}{dw^{(v)}} = d\varphi^{(u)}(g_i^{(u)}) \frac{dg_i^{(u)}}{dw^{(v)}}.$$

$$\frac{dg_i^{(u)}}{dw^{(v)}} = \begin{cases} S^{(u)}(f_i^{(u-1)}; \cdot) & \text{si } u = v, \\ S^{(u)} \left(\frac{df_i^{(u-1)}}{dw^{(v)}}; w^{(u)} \right) & \text{si } v > u, \\ 0 & \text{si } v < u. \end{cases}$$

Phénomène de disparition du gradient.

$$u = 1, \dots, U, \quad \left| \frac{df_i^{(u)}}{dw^{(v)}} \right| \leq \left| d\varphi^{(u)}(g_i^{(u)}) \right| \left| \frac{df_i^{(u-1)}}{dw^{(v)}} \right| |w^{(u)}|.$$

- La norme de la différentielle n'est pas du même ordre à toutes les couches.
- Certaines fonctions d'activations $\varphi^{(u)}$ peuvent avoir un effet saturant et annuler les différentielles ($|d\varphi^{(u)}(g_i^{(u)})| \simeq 0$).
- Conséquence : Apprentissage très lent des couches hautes.

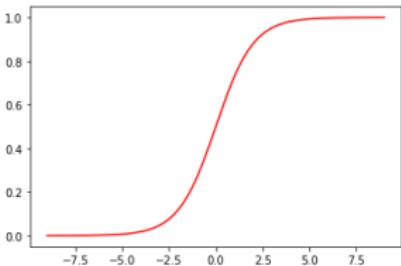
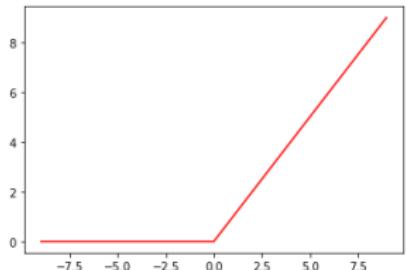


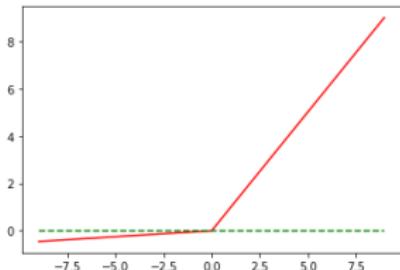
Figure – Effet saturant de la fonction d'activation logistique

Fonctions d'activation non saturantes.



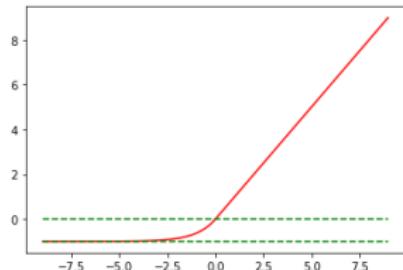
ReLU,
Glorot et Bengio,
2010

$$\varphi(t) = \max(0, t)$$



leaky ReLU(α),
ex. $\alpha = 0.01$

$$\varphi(t) = \max(\alpha t, t)$$



ELU(α)
Clevert et al., 2015

$$\varphi(t) = \alpha(\exp(t) - 1) \quad \text{si } t < 0 \text{ et } t \text{ sinon}$$

Normalisation par lot (Batch normalisation).

- Problème de l'"internal covariate shift" : statistiques des sorties d'une couche varient en fonction de la couche précédente.
- Solution (Ioffe et Szegedy, 2015) : centrage et normalisation des sorties de chaque couche de convolution (ou dense).

Sorties de la u ème couche de convolution :

$$g_i^{(u)} = S^{(u)}(f_i^{(u-1)}; w^{(u)}).$$

Estimation de l'espérance et de la variance de $g^{(u)}$ sur un lot B :

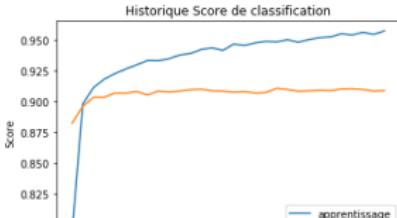
$$\hat{\mu}_B^{(u)} = \frac{1}{|B|} \sum_{i \in B} g_i^{(u)} \quad \text{et} \quad \hat{v}_B^{(u)} = \frac{1}{|B|} \sum_{i \in B} (g_i^{(u)} - \hat{\mu}_B^{(u)})^2.$$

Centrage et normalisation des sorties :

$$\tilde{g}_i^{(u)} = \frac{1}{\sqrt{\hat{v}_B^{(u)}} + \epsilon} (g_i^{(u)} - \hat{\mu}_B^{(u)}).$$

Problème du surajustement

- Sur-ajustement : le modèle appris est bien ajusté aux données d'apprentissage mais donne des prévisions médiocre lorsqu'appliqué à des données nouvelles.
- Problème statistique qui vient du fait que la complexité du modèle (nombre de paramètres) est trop grand par rapport aux données d'apprentissage.
- Le sur-ajustement peut se détecter lors de l'optimisation sur des données de validation.
- Il se diagnostique également après l'optimisation sur des données de test.



Régularisation.

- Pénalisation du problème d'optimisation :

$$\tilde{E}_\lambda(w) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \lambda R(w),$$

où $R(w)$ pénalise les grandes valeurs de w (par exemple, $R(w) = |w|_2^2$ ou $R(w) = |w|_1$).

- En contrignant les solutions, la pénalisation réduit la complexité du modèle et peut atténuer le sur-ajustement.
- Elle permet aussi de mieux poser le problème d'optimisation en termes d'existence et de stabilité des solutions.
- En keras, les options de régularisation se mettent sur les couches.

Dropout.

- Hinton, 2012 et Srivastava et al. 2014.
- Principe : Au cours de l'optimisation, on gèle aléatoirement des sorties de couches pour se concentrer sur l'apprentissage d'un sous-modèle (sous-graphe) du réseau.
- Sur des couches u (sauf la dernière), on tire un échantillon i.i.d $b^{(u)}$ de loi de Bernoulli $\mathcal{B}(p)$ de même dimension que les sorties $g_i^{(u)}$. Puis on effectue les opérations :

$$\begin{cases} g_i^{(u)} &= S^{(u)}(f_i^{(u-1)}; w^{(u)}), \\ \tilde{g}_i^{(u)} &= g_i^{(u)} \otimes b^{(u)}, \\ f_i^{(u)} &= \varphi^{(u)}(\tilde{g}_i^{(u)}). \end{cases}$$

où \otimes est une produit point par point.

Autres techniques pour atténuer le sur-ajustement.

- Augmentation des données : ajouter des transformations des données d'apprentissage (translation, rotation, changement de contraste,...), le plus souvent à la volée pendant l'apprentissage (pour éviter d'utiliser trop de stockage).
- Ré-utilisation des couches convolutionnelles (représentation de l'image) d'un réseau déjà appris sur une base d'images pour classification d'images d'une autre base (en général, plus petite). Fine-tuning lorsque modification de certaines couches convolutionnelles lors de l'apprentissage du nouveau classifieur.

Optimisation.

- Fonction à minimiser : $E_n(w) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, \hat{y}_i(w)).$
- Descente de gradient : une itération t

$$w(t+1) = w(t) - \frac{\rho_t}{n} \sum_{i=1}^n \nabla \ell(y_i, \hat{y}_i(w)),$$

- Descente de gradient stochastique par mini-lots (batchs). Une époque (epochs) $e + 1$:
 - Initialisation : $\tilde{w}(1) = w(e)$ puis pour les itérations $t = 1, \dots, T - 1$,

$$\tilde{w}(t+1) = \tilde{w}(t) - \frac{\rho_e}{m} \sum_{i \in B_w(t)} \nabla \ell(y_i, \hat{y}_i(\tilde{w}(t))),$$

où, à t fixé, les $B_u(t)$ sont des ensembles disjoints de m indices tirés aléatoirement dans $\{1, \dots, n\}$.

- $w(e+1) = \tilde{w}(T)$



Accélérations de la descente de gradient.

- Descente de gradient : $w(t + 1) = w(t) - \rho C(w)$.
- Descente de gradient avec inertie : On garde une trace des gradients précédents et on l'utilise pour la mise à jour de w :

$$\begin{aligned} g(t + 1) &= \beta g(t) + (1 - \beta) \nabla C(w(t)), \\ w(t + 1) &= w(t) - \rho g(t + 1), \end{aligned}$$

avec $\beta \in (0, 1)$ paramètre réglant l'importance accordée aux gradients antérieurs.

- Inertie de Nesterov :

$$\begin{aligned} g(t + 1) &= \beta g(t) + (1 - \beta) \nabla C(w(t) - \rho \beta g(t)), \\ w(t + 1) &= w(t) - \rho g(t + 1). \end{aligned}$$

AdaGrad, RMSProp, Adam.

- AdaGrad :

$$\begin{aligned} v(t+1) &= v(t) + \nabla C(w(t)) \otimes \nabla C(w(t)), \\ w(t+1) &= w(t) - \rho \nabla C(w(t)) \oslash \sqrt{v(t+1) + \epsilon}, \end{aligned}$$

où \otimes et \oslash sont les multiplications et divisions point à point.

- RMSProp :

$$\begin{aligned} v(t+1) &= \beta v(t) + (1 - \beta) \nabla C(w(t)) \otimes \nabla C(w(t)), \\ w(t+1) &= w(t) - \rho \nabla C(w(t)) \oslash \sqrt{v(t+1) + \epsilon}, \end{aligned}$$

avec $\beta \in (0, 1)$.

- Adam :

$$\begin{aligned} g(t+1) &= \beta_1 g(t) + (1 - \beta_1) \nabla C(w(t)), \\ v(t+1) &= \beta_2 v(t) + (1 - \beta_2) \nabla C(w(t)) \otimes \nabla C(w(t)), \\ w(t+1) &= w(t) - \rho g(t+1) \oslash \sqrt{v(t+1) + \epsilon}, \end{aligned}$$

avec $\beta_1, \beta_2 \in (0, 1)$